## CONTENTS

1. Getting started .................................................. 3
2. Authors ......................................................... 5
Python Module Index ........................................... 51
Index ............................................................. 53
astir is a modelling framework for the assignment of cell type across a range of single-cell technologies such as Imaging Mass Cytometry (IMC). astir is built using pytorch and uses recognition networks for fast minibatch stochastic variational inference.

Key applications:

- Automated assignment of cell type and state from highly multiplexed imaging and proteomic data
- Diagnostic measures to check quality of resulting type and state inferences
- Ability to map new data to cell types and states trained on existing data using recognition neural networks
- A range of plotting and data loading utilities
CHAPTER
ONE

GETTING STARTED

Launch the interactive tutorial:
See the full documentation and check out the tutorials.
2.1 Installation

2.1.1 Prerequisites
Install python 3.7. Astir uses python 3.*

2.1.2 Astir installation

PyPI

```
pip3 install astir
```

Dev

Clone this repo and run

```
git clone https://github.com/camlab-bioml/astir.git
cd astir
pip install -e .
```

2.2 Tutorials

2.2.1 Getting started with astir

Table of Contents:

- 0 Loading necessary libraries
- 1 Load data
- 2 Fitting cell type
0. Load necessary libraries

```python
[# !pip install -e ../../../..

import os
import sys
module_path = os.path.abspath(os.path.join('..', '..'))
if module_path not in sys.path:
    sys.path.append(module_path)
]
```

```python
[5]:

```python
from astir.data import from_csv_yaml
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%load_ext autoreload
%autoreload 2
%matplotlib inline
```

1. Load data

We start by reading expression data in the form of a csv file and marker gene information in the form of a yaml file:

```python
[1]: expression_mat_path = "../../../tests/test-data/sce.csv"
# expression_mat_path = "data/sample_data.csv"
yaml_marker_path = "../../../tests/test-data/jackson-2020-markers.yml"
```

**Note:** Expression data should already be in a cleaned and normalized form. For IMC data, we perform this by an arcsinh transformation of the data with a cofactor so `transformed_expression = archsinh(raw_expression / cofactor)`, where typically `cofactor=5`.

We can view both the expression data and marker data:

```bash
[2]: !head -n 20 ../../../tests/test-data/jackson-2020-markers.yml
```

```
cell_states:
    RTK_signalling:
```

(continues on next page)
- Her2
- EGFR
proliferation:
  - Ki-67
  - phospho Histone
mTOR_signalling:
  - phospho mTOR
  - phospho S6
apoptosis:
  - cleaved PARP
  - Cleaved Caspase3
cell_types:
  - stromal:
    - Vimentin
    - Fibronectin
  - B cells:

[6]: pd.read_csv(expression_mat_path, index_col=0)['[EGFR', 'E-Cadherin', 'CD45', 'Cytokeratin_ ...
...5']].head()

[6]:

| BaselTMA_SP41_186_X5Y4_3679 | 0.346787 | 0.938354 | 0.227730 | 0.095283 |
| BaselTMA_SP41_153_X7Y5_246 | 0.833752 | 1.364884 | 0.068526 | 0.124031 |
| BaselTMA_SP41_20_X12Y5_197 | 0.110006 | 0.177361 | 0.301222 | 0.052750 |
| BaselTMA_SP41_14_X1Y8_84  | 0.282666 | 1.122174 | 0.606941 | 0.093352 |
| BaselTMA_SP41_166_X15Y4_266 | 0.209066 | 0.402554 | 0.588273 | 0.064545 |

Then we can create an astir object using the from_csv_yaml function. For more data loading options, see the data loading tutorial.

[7]: ast = from_csv_yaml(expression_mat_path, marker_yam=1=yam_marker_path)
print(ast)

Astir object, 6 cell types, 4 cell states, 100 cells

2. Fitting cell types

To fit cell types, simply call

[8]: ast.fit_type(max_epochs=10, n_init=3, n_init_epochs=2)

training restart 1/3: 100%|| 2/2 [ 4.51epochs/s, current loss: 745.5]
training restart 2/3: 100%|| 2/2 [108.41epochs/s, current loss: 776.6]
training restart 3/3: 100%|| 2/2 [40.70epochs/s, current loss: 774.7]
training restart (final): 100%|| 10/10 [82.58epochs/s, current loss: 709.0]

Note: Controlling inference There are many different options for controlling inference in the fit_type function, including max_epochs (maximum number of epochs to train), learning_rate (ADAM optimizer learning rate),
 batch_size (minibatch size), delta_loss (stops iteration once the change in loss falls below this value), n_inits (number of restarts using random initializations). For full details, see the function documentation.

We should always plot the losses to assess convergence:

```python
[9]: plt.figure(figsize=(5,4))
plt.plot(np.arange(len(ast.get_type_losses())), ast.get_type_losses())
plt.ylabel("Loss")
plt.xlabel("Epoch")
```

![Graph showing loss over epochs](image)

We can then get cell type assignment probabilities by calling

```python
[10]: assignments = ast.get_celltype_probabilities()
assignments
```

<table>
<thead>
<tr>
<th>Sample</th>
<th>Stromal</th>
<th>B Cells</th>
<th>T Cells</th>
<th>Macrophage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>0.088444</td>
<td>0.084654</td>
<td>0.198811</td>
<td>0.105010</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>0.110921</td>
<td>0.156938</td>
<td>0.193806</td>
<td>0.122539</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12Y5_197</td>
<td>0.099029</td>
<td>0.107530</td>
<td>0.210399</td>
<td>0.135788</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>0.117755</td>
<td>0.119934</td>
<td>0.221191</td>
<td>0.103741</td>
</tr>
<tr>
<td>BaselTMA_SP41_166_X15Y4_266</td>
<td>0.102338</td>
<td>0.106135</td>
<td>0.221755</td>
<td>0.135476</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>BaselTMA_SP41_114_X13Y4_1057</td>
<td>0.100268</td>
<td>0.128181</td>
<td>0.204113</td>
<td>0.140633</td>
</tr>
<tr>
<td>BaselTMA_SP41_141_X11Y2_2596</td>
<td>0.112603</td>
<td>0.143363</td>
<td>0.201669</td>
<td>0.148209</td>
</tr>
<tr>
<td>BaselTMA_SP41_100_X15Y5_170</td>
<td>0.111955</td>
<td>0.130958</td>
<td>0.203715</td>
<td>0.144000</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_2604</td>
<td>0.076987</td>
<td>0.084889</td>
<td>0.218953</td>
<td>0.114034</td>
</tr>
<tr>
<td>BaselTMA_SP41_186_X5Y4_81</td>
<td>0.131032</td>
<td>0.128943</td>
<td>0.206584</td>
<td>0.123474</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sample</th>
<th>Epithelial (Basal)</th>
<th>Epithelial (Luminal)</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>0.199379</td>
<td>0.119610</td>
<td>0.204091</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>0.200939</td>
<td>0.114491</td>
<td>0.100367</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12Y5_197</td>
<td>0.174190</td>
<td>0.126138</td>
<td>0.146926</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>0.150916</td>
<td>0.140125</td>
<td>0.146337</td>
</tr>
<tr>
<td>BaselTMA_SP41_166_X15Y4_266</td>
<td>0.157753</td>
<td>0.125728</td>
<td>0.150815</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

(continues on next page)
We can also visualize the assignment probabilities using a heatmap:

```{.ipython}
[11]: sns.heatmap(assignments)
```

where each row corresponds to a cell, and each column to a cell type, with the entry being the probability of that cell belonging to a particular cell type.

To fetch an array corresponding to the most likely cell type assignments, call

```{.ipython}
[12]: ast.get_celltypes()
```

(continues on next page)
It is important to run diagnostics to ensure that cell types express their markers at higher levels than other cell types. To do this, run the `diagnostics_celltype()` function, which will alert to any issues if a cell type doesn’t express its marker significantly higher than an alternative cell type (for which that protein isn’t a marker):

```python
[12]: ast.diagnostics_celltype().head(n=10)
[12]: Empty DataFrame
Columns: [feature, should be expressed higher in, than, mean cell type 1, mean cell type 2, p-value, note]
Index: []
```

**Note:** In this tutorial, we end up with many “Only 1 cell in a type: comparison not possible” notes - this is simply because the small dataset size results in only a single cell assigned to many types, making statistical testing infeasible.

Calling `ast.diagnostics_celltype()` returns a `pd.DataFrame`, where each column corresponds to a particular protein and two cell types, with a warning if the protein is not expressed at higher levels in the cell type for which it is a marker than the cell type for which it is not.

The diagnostics:

1. Iterates through every cell type and every marker for that cell type
2. Given a cell type \( c \) and marker \( g \), find the set of cell types \( D \) that don’t have \( g \) as a marker
3. For each cell type \( d \) in \( D \), perform a t-test between the expression of marker \( g \) in \( c \) vs \( d \)
4. If \( g \) is not expressed significantly higher (at significance \( \alpha \) ), output a diagnostic explaining this for further investigation.

If multiple issues are found, the markers and cell types may need refined.

### 3. Fitting cell state

**Caution:** Cell state fitting in Astir is currently experimental and not included in the initial paper.

Similarly as before, to fit cell state, call

```python
[45]: ast.fit_state(batch_size = 1024, learning_rate=1e-3, max_epochs=10)
/Users/jinelles.h/Documents/Camlab/astir-top-level/astir/astir.py:222: UserWarning:
    Delta loss batch size is greater than the number of epochs
warnings.warn("Delta loss batch size is greater than the number of epochs")
training restart 1/5: 100%|| 5/5 [129.59epochs/s, current loss: 196.5]
training restart 2/5: 100%|| 5/5 [117.63epochs/s, current loss: 231.9]
training restart 3/5: 100%|| 5/5 [124.46epochs/s, current loss: 217.7]
```

(continues on next page)
and similarly plot the losses via

```python
[14]: plt.figure(figsize=(5,4))
plt.plot(np.arange(len(ast.get_state_losses())), ast.get_state_losses())
plt.ylabel("Loss")
plt.xlabel("Epoch")
```

and cell state assignments can be inferred via

```python
[15]: states = ast.get_cellstates()

<table>
<thead>
<tr>
<th></th>
<th>RTK_signalling</th>
<th>proliferation</th>
<th>mTOR_signalling</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>0.568386</td>
<td>0.900861</td>
<td>0.656078</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>0.421357</td>
<td>0.000000</td>
<td>0.524859</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12Y5_197</td>
<td>0.944829</td>
<td>0.561159</td>
<td>0.979277</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>0.858426</td>
<td>0.705787</td>
<td>0.938068</td>
</tr>
<tr>
<td>BaselTMA_SP41_166_X15Y4_266</td>
<td>0.933672</td>
<td>0.574031</td>
<td>0.980568</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>BaselTMA_SP41_114_X13Y4_1057</td>
<td>0.881551</td>
<td>0.447002</td>
<td>0.899008</td>
</tr>
<tr>
<td>BaselTMA_SP41_141_X11Y2_2596</td>
<td>0.767853</td>
<td>0.684847</td>
<td>0.856773</td>
</tr>
<tr>
<td>BaselTMA_SP41_100_X15Y5_170</td>
<td>0.952977</td>
<td>0.548220</td>
<td>0.977899</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_2604</td>
<td>0.836241</td>
<td>0.692617</td>
<td>0.908256</td>
</tr>
<tr>
<td>BaselTMA_SP41_186_X5Y4_81</td>
<td>0.691698</td>
<td>0.719179</td>
<td>0.705012</td>
</tr>
</tbody>
</table>

apoptosis

BaselineTMA_SP41_186_X5Y4_3679 0.548110
Cell state diagnostics

It is important to run diagnostics on cell states model for the same reasons stated for the cell type model. Astir. `diagnostics_cellstate()` spots any non marker protein and pathway pairs whose expressions are higher than those of the marker proteins of the pathway.

```python
[17]: ast.diagnostics_cellstate().head(n=10)
```

<table>
<thead>
<tr>
<th>pathway</th>
<th>protein A</th>
<th>correlation of protein A</th>
<th>protein B</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTK_signalling</td>
<td>Her2</td>
<td>-0.542371</td>
<td>Ki-67</td>
</tr>
<tr>
<td>proliferation</td>
<td>Ki-67</td>
<td>0.241679</td>
<td>Her2</td>
</tr>
<tr>
<td>proliferation</td>
<td>Ki-67</td>
<td>0.241679</td>
<td>phospho S6</td>
</tr>
<tr>
<td>proliferation</td>
<td>Ki-67</td>
<td>0.241679</td>
<td>phospho mTOR</td>
</tr>
<tr>
<td>mTOR_signalling</td>
<td>phospho mTOR</td>
<td>-0.776636</td>
<td>Cleaved Caspase3</td>
</tr>
</tbody>
</table>
```

(continues on next page)

(continues from previous page)
 Calling ast.diagnostics_cellstate() returns a pd.DataFrame, where each column corresponds to a particular protein and two cell types, with a warning if the protein is not expressed at higher levels in the cell state for which it is a marker than the cell state for which it is not.

The diagnostics:

1. Get correlations between all cell states and proteins
2. For each cell state c, get the smallest correlation with marker g
3. For each cell state c and its non marker g, find any correlation that is bigger than those smallest correlation for c.
4. Any c and g pairs found in step 3 will be included in the output of Astir.diagnostics_cellstate(), including an explanation.

If multiple issues are found, the markers and cell states may need refined.

4. Saving results

Both cell type and cell state information can easily be saved to disk via

```python
[18]: ast.type_to_csv("data/cell-types.csv")
ast.state_to_csv("data/cell-states.csv")
```

```bash
[19]: !head -n 3 data/cell-types.csv
```

<table>
<thead>
<tr>
<th>cell_type</th>
<th>BaselTMA_SP41_186_X5Y4_3679</th>
<th>BaselTMA_SP41_153_X7Y5_246</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td>0.5683861877028941</td>
<td>0.656078413193121</td>
</tr>
</tbody>
</table>

```bash
[20]: !head -n 3 data/cell-states.csv
```

<table>
<thead>
<tr>
<th>RTK_signalling,proliferation,mTOR_signalling,apoptosis</th>
<th>BaselTMA_SP41_186_X5Y4_3679</th>
<th>BaselTMA_SP41_153_X7Y5_246</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5683861877028941</td>
<td>0.42135746208636826</td>
<td></td>
</tr>
</tbody>
</table>

where the first (unnamed) column always corresponds to the cell name/ID.
5. Accessing internal functions and data

Data stored in `astir` objects is in the form of an SCDataSet. These can be retrieved via

[21]:
celltype_data = ast.get_type_dataset()
celltype_data

[21]: <astir.data.scdataset.SCDataset at 0x7f8041983290>

and similarly for cell state via `ast.get_state_dataset()`.

These have several helper functions to retrieve relevant information to the dataset:

[22]:
celltype_data.get_cell_names()[0:4] # cell names

[22]: ['BaselTMA_SP41_186_X5Y4_3679',
     'BaselTMA_SP41_153_X7Y5_246',
     'BaselTMA_SP41_20_X12Y5_197',
     'BaselTMA_SP41_14_X1Y8_84']

[23]:
celltype_data.get_classes() # cell type names

[23]: ['stromal',
     'B cells',
     'T cells',
     'macrophage',
     'epithelial(basal)',
     'epithelial(luminal)']

[24]:
print(celltype_data.get_n_classes()) # number of cell types
print(celltype_data.get_n_features()) # number of features / proteins

6
14

[25]:
celltype_data.get_exprs() # Return a torch tensor corresponding to the expression data

[25]: tensor([[0.1026, 0.1004, 0.2277, ..., 0.6097, 2.227252],
         [0.1081, 0.0176, 0.0685, ..., 1.0622, 0.5026, 3.9632],
         [0.0498, 0.0943, 0.3012, ..., 0.1601, 0.8102, 0.0481],
         ...,
         [0.0695, 0.0119, 0.0869, ..., 0.4487, 0.7593, 1.4923],
         [0.0929, 0.1266, 0.2395, ..., 0.4405, 2.2464, 0.4174],
         [0.0618, 0.1439, 0.2476, ..., 0.7055, 3.1238, 0.2552]],
        dtype=torch.float64)

dtype=torch.float64)

[26]:
celltype_data.get_exprs_df() # Return a pandas DataFrame corresponding to the expression data

[26]:
<table>
<thead>
<tr>
<th>CD20</th>
<th>CD3</th>
<th>CD45</th>
<th>CD68</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>0.100256</td>
<td>0.1000401</td>
<td>0.227730</td>
<td>2.227252</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>0.108137</td>
<td>0.017637</td>
<td>0.068526</td>
<td>0.208297</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12Y5_197</td>
<td>0.049809</td>
<td>0.094316</td>
<td>0.301222</td>
<td>0.51624</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>0.024256</td>
<td>0.140441</td>
<td>0.606941</td>
<td>0.490982</td>
</tr>
<tr>
<td>BaselTMA_SP41_166_X15Y4_266</td>
<td>0.138571</td>
<td>0.111722</td>
<td>0.588273</td>
<td>1.039967</td>
</tr>
</tbody>
</table>

(continues on next page)
<table>
<thead>
<tr>
<th></th>
<th>Cytokeratin 14</th>
<th>Cytokeratin 19</th>
<th>Cytokeratin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>0.195163</td>
<td>0.190923</td>
<td>0.095283</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>0.234853</td>
<td>0.685858</td>
<td>0.124031</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12Y5_197</td>
<td>0.072666</td>
<td>0.115979</td>
<td>0.052750</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>0.165863</td>
<td>0.652143</td>
<td>0.093352</td>
</tr>
<tr>
<td>BaselTMA_SP41_166_X15Y4_266</td>
<td>0.162696</td>
<td>0.086235</td>
<td>0.064545</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Cytokeratin 7</th>
<th>Cytokeratin 8/18</th>
<th>E-Cadherin</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>0.057050</td>
<td>0.461040</td>
<td>0.938354</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>0.485330</td>
<td>0.382767</td>
<td>1.364884</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12Y5_197</td>
<td>0.035875</td>
<td>0.094383</td>
<td>1.122174</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>0.009627</td>
<td>0.046967</td>
<td>0.402554</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Fibronectin</th>
<th>Her2</th>
<th>Vimentin</th>
<th>pan Cytokeratin</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>1.829905</td>
<td>0.609694</td>
<td>2.215089</td>
<td>0.771352</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>1.226994</td>
<td>1.062229</td>
<td>0.502627</td>
<td>3.963248</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12Y5_197</td>
<td>2.222520</td>
<td>0.160135</td>
<td>0.810243</td>
<td>0.948100</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>1.402750</td>
<td>1.133448</td>
<td>1.742495</td>
<td>1.917118</td>
</tr>
<tr>
<td>BaselTMA_SP41_166_X15Y4_266</td>
<td>2.669947</td>
<td>0.558439</td>
<td>1.659587</td>
<td>0.687005</td>
</tr>
</tbody>
</table>

[100 rows x 14 columns]
<table>
<thead>
<tr>
<th>TMA</th>
<th>CD20</th>
<th>CD3</th>
<th>CD45</th>
<th>CD68</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>0.020514</td>
<td>0.020079</td>
<td>0.045530</td>
<td>0.384408</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>0.021626</td>
<td>0.003527</td>
<td>0.013705</td>
<td>0.041647</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12YS_197</td>
<td>0.009962</td>
<td>0.018862</td>
<td>0.060208</td>
<td>0.116064</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>0.004851</td>
<td>0.028085</td>
<td>0.121092</td>
<td>0.098039</td>
</tr>
<tr>
<td>BaselTMA_SP41_166_X15Y4_266</td>
<td>0.027711</td>
<td>0.022343</td>
<td>0.117385</td>
<td>0.206522</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMA</th>
<th>Cytokeratin 14</th>
<th>Cytokeratin 19</th>
<th>Cytokeratin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>0.039023</td>
<td>0.038175</td>
<td>0.019055</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>0.046953</td>
<td>0.136745</td>
<td>0.024804</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12YS_197</td>
<td>0.014533</td>
<td>0.023194</td>
<td>0.010550</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>0.033167</td>
<td>0.130062</td>
<td>0.018669</td>
</tr>
<tr>
<td>BaselTMA_SP41_166_X15Y4_266</td>
<td>0.032533</td>
<td>0.017246</td>
<td>0.012909</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMA</th>
<th>Cytokeratin 7</th>
<th>Cytokeratin 8/18</th>
<th>E-Cadherin</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>0.011410</td>
<td>0.092078</td>
<td>0.186586</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>0.096914</td>
<td>0.076479</td>
<td>0.269695</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12YS_197</td>
<td>0.007175</td>
<td>0.004058</td>
<td>0.035465</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>0.070282</td>
<td>0.179905</td>
<td>0.225292</td>
</tr>
<tr>
<td>BaselTMA_SP41_166_X15Y4_266</td>
<td>0.001925</td>
<td>0.009393</td>
<td>0.080424</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMA</th>
<th>Fibronectin</th>
<th>Her2</th>
<th>Vimentin</th>
<th>pan Cytokeratin</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_186_X5Y4_3679</td>
<td>0.358267</td>
<td>0.121639</td>
<td>0.429674</td>
<td>0.153665</td>
</tr>
<tr>
<td>BaselTMA_SP41_153_X7Y5_246</td>
<td>0.243000</td>
<td>0.210879</td>
<td>0.100357</td>
<td>0.726918</td>
</tr>
<tr>
<td>BaselTMA_SP41_20_X12YS_197</td>
<td>0.431033</td>
<td>0.032021</td>
<td>0.161348</td>
<td>0.009620</td>
</tr>
<tr>
<td>BaselTMA_SP41_14_X1Y8_84</td>
<td>0.276994</td>
<td>0.224792</td>
<td>0.341804</td>
<td>0.374601</td>
</tr>
<tr>
<td>BaselTMA_SP41_166_X15Y4_266</td>
<td>0.511404</td>
<td>0.111457</td>
<td>0.326107</td>
<td>0.136972</td>
</tr>
</tbody>
</table>

[100 rows x 14 columns]
6. Saving models

After fixing the models, we can save the cell type/state assignment, the losses, the parameters (e.g. \( \mu, \rho, \log\sigma, \) etc) and the run informations (e.g. batch size, learning rate, delta loss, etc) to an hdf5 file.

```
[29]: ast.save_models("data/astir_summary.hdf5")
```

The hierarchy of the hdf5 file would be:

```
<table>
<thead>
<tr>
<th>/celltype_model</th>
</tr>
</thead>
<tbody>
<tr>
<td>/celltype_assignments</td>
</tr>
<tr>
<td>/run_info</td>
</tr>
<tr>
<td>/parameters</td>
</tr>
<tr>
<td>/losses</td>
</tr>
<tr>
<td>/recog_net</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>/cellstate_model</th>
</tr>
</thead>
<tbody>
<tr>
<td>/losses</td>
</tr>
<tr>
<td>/parameters</td>
</tr>
<tr>
<td>/run_info</td>
</tr>
<tr>
<td>/cellstate_assignments</td>
</tr>
</tbody>
</table>
```

Only the model that is trained will be saved (CellTypeModel or CellStateModel or both). If the function is called before any model is trained, exception will be raised. Data saved in the file is either int or np.array.

7. Plot clustermap of expression data

After fixing the cell type model, we can also plot a heatmap of protein expression of cells clustered by type. The heatmap will be saved at the location plot_name, which is default to ".celltype_protein_cluster.png"

```
[30]: ast.type_clustermap(plot_name="./img/celltype_protein_cluster.png", threshold = 0.7, figsize=(7, 5))
```
Note: threshold is the probability threshold above which a cell is assigned to a cell type, default to 0.7.

8. Hierarchical model specification

In the marker yaml file, the user can also add a section called hierarchy, which specifies the hierarchical structure of cell types. Here’s an example:

```yaml
hierarchy:
  epithelial_cells:
    - epithelial(luminal)
    - epithelial(basal)
  immune_cells:
    non-lymphocytes:
      - macrophage
    lymphocytes:
      - T cells
      - B cells
```

Some notes: 1. The section would be accessed by key hierarchy. 2. In the section, the higher-levelled cell type names should be the keys. 3. The values in the section should also exist as the cell type names in the cell_types section. (e.g. if we have "B cells" in marker["hierarchy"]["immune"], we should also be able to get marker["cell_types"]["B cells"]) 4. In terms of depth in the example: - depth=1: assign to epithelial_cells or immune_cells - depth=2: assign to epithelial(luminal), epithelial(basal), non-lymphocyte and lymphocyte - depth=3: assign to epithelial(luminal), epithelial(basal), macrophage, T cells and B cells

This section could be used to summarize the cell types assignment at a higher hierarchical level. (e.g. a cell is predicted as “immune” instead of “B cells” or “T cells”)

```python
hierarchy_probs = ast.assign_celltype_hierarchy(depth = 1)
hierarchy_probs.head()
```
To make it more clear, here’s a heatmap for the cell assignment in a higher hierarchy:
The way it is calculated is simply summing up the probabilities of the cell type assignments under the same hierarchy.

9. Using astir as command line tool

astir could also be used as a command line tool with csv input. Here are some example.

To fit cell types on the sample csv file and marker with a design matrix, setting n_init to 3 and batch_size to 128:

```
!astir type ../../../tests/test-data/test_data.csv ../../../tests/test-data/jackson-2020-markers.yml data/test_data_type_assignments.csv --design ../../../tests/test-data/design.csv --n_init 3 --batch_size 128
```

```
training restart 1/3: 100%|| 5/5 [164.98epochs/s, current loss: -25.6]
training restart 2/3: 100%|| 5/5 [181.48epochs/s, current loss: -33.4]
training restart 3/3: 100%|| 5/5 [177.48epochs/s, current loss: -12.5]
training restart (final): 100%|| 50/50 [198.48epochs/s, current loss: -591.9]
Maximum epochs reached. More iteration may be needed to complete the training.
warnings.warn(msg)
```

To fit cell states on the sample csv file and marker, setting learning_rate to 5e-4, dropout_rate to 0.2 and batch_norm to True:

```
!astir state ../../../tests/test-data/test_data.csv ../../../tests/test-data/jackson-2020-markers.yml data/test_data_type_assignments.csv --design ../../../tests/test-data/design.csv --learning_rate 5e-4 --dropout_rate 0.2 --batch_norm True
```

(continues on next page)
Moreover, astir could also be used as a converter which converts rds file with SingleCellExperiment to csv file. See more details

Run astir -h, astir type -h, astir state -h and astir convert -h in the command line for more details.

\[33\]:

### 2.2.2 Loading data into astir

#### Table of Contents:

- 0 Loading necessary libraries
- 1 Starting Astir within python
  - 1.0 Loading marker dictionary and design matrix
  - 1.1 Loading data as pd.DataFrame
  - 1.2 Loading data as np.array
  - 1.3 Loading data as SCDataset
- 2 Loading from csv and yaml files
- 3 Loading from a directory of csvs and yaml
- 4 Loading from loom
- 5 Loading from anndata
- 6 Loading model from hdf5
- 7 Converting rds file (with data as SingleCellExperiment) to csv file

#### 0. Loading necessary libraries and define paths

```
[1]: # !pip install -e ../../..
    import os
    import sys

    module_path = os.path.abspath(os.path.join('../../..'))
    if module_path not in sys.path:
        sys.path.append(module_path)
    module_path = os.path.abspath(os.path.join('../../../astir'))
    if module_path not in sys.path:
        sys.path.append(module_path)
    print(sys.path)
```

1. Starting Astir within python

The input dataset should represent protein expression in single cells. The rows should represent each cell (one row per cell) and the columns should represent each protein (one column per protein). A marker which maps the features (proteins) to cell type/state may be required. A design matrix is optional. If provided, it should be either np.array or pd.DataFrame.

The initialization of Astir requires input dataset `input_expr` as one of pd.DataFrame, Tuple[np.array, List[str], List[str]] and Tuple[SCDataset, SCDataset].

Note: `dtype` and `random_seed` are always customizable. `dtype` is default to `torch.float64` and `random_seed` is default to 1234.

1.0 Loading marker dictionary and design matrix

Marker Dictionary

```python
with open(yaml_marker_path, "r") as stream:
    marker_dict = yaml.safe_load(stream)
print(marker_dict)
```

```yaml
yaml_marker_path = "../../../tests/test-data/jackson-2020-markers.yml"
```

```python
import yaml
import pandas as pd
import astir as ast
import numpy as np
import torch

[2]:
import yaml
import pandas as pd
import astir as ast
import numpy as np
import torch

[3]:
yaml_marker_path = "../../../tests/test-data/jackson-2020-markers.yml"
design_mat_path = "../../../tests/test-data/design.csv"
expression_mat_path = "../../../tests/test-data/test_data.csv"
expression_dir_path = "../../../tests/test-data/test-dir-read"
expression_loom_path = "../../../tests/test-data/basel_100.loom"
expression_anndata_path="../../../tests/test-data/adata_small.h5ad"

[4]:
with open(yaml_marker_path, "r") as stream:
    marker_dict = yaml.safe_load(stream)
print(marker_dict)

{'cell_states': {'RTK_signalling': ['Her2', 'EGFR'], 'proliferation': ['Ki-67', 'phospho_Histone'], 'mTOR_signalling': ['phospho mTOR', 'phospho S6'], 'apoptosis': ['cleaved_PARP', 'Cleaved Caspase3']}, 'cell_types': {'stromal': ['Vimentin', 'Fibronectin'], 'B_cells': ['CD45', 'CD20'], 'T_cells': ['CD45', 'CD3'], 'macrophage': ['CD45', 'CD68'], 'epithelial(basal)': ['E-Cadherin', 'pan Cytokeratin', 'Cytokeratin 5', 'Cytokeratin 14', 'Her2'], 'epithelial(luminal)': ['E-Cadherin', 'pan Cytokeratin', 'Cytokeratin 7', 'Cytokeratin 8/18', 'Cytokeratin 19', 'Cytokeratin 5', 'Her2'], 'hierachy': {'epithelial_cells': ['E-Cadherin', 'luminal epithelial', 'basal epithelial'], 'immune_cells': {'T_cells': 'null', 'B_cells': 'null'}}
```
Some notes:

1. The marker `marker_dict` is not required when `input_expr` is `Tuple[SCDataset, SCDataset]`. Otherwise, it is required to be `Dict[str, Dict[str, str]]`.

2. The outer dictionary may have at most three keys: `cell_type`, `cell_state` and `hierarchy`. `cell_type` and `cell_state` maps to the corresponding dictionary which maps the name of cell type/state to protein features. `hierarchy` maps to the dictionary which indicates the cell type hierarchy.

3. If the user is only intended to classify one of cell type and cell state, only the intended marker dictionary should be provided. So that `marker_dict` is one of `{"cell_state": {...}, "{cell_type": {...}}` and `{"cell_type": {...}, "cell_state": {...}}`.

4. The `hierarchy` dictionary should be included when the client tends to call Astir. `assign_celltype_hierarchy()`

**Design matrix:**

Note that the design matrix must have the same number of rows as there are number of cells.

```python
[5]: design_df = pd.read_csv(design_mat_path, index_col=0)
print(design_df.shape)
```

```
(49, 40)
```

Note: `design` is not necessary when `input_expr` is `Tuple[SCDataset, SCDataset]`. Otherwise it is optional.

### 1.1 Loading data as `pd.DataFrame`

When the input is `pd.DataFrame`, its row and column should respectively represent the cells and the features (proteins).

```python
[6]: df_expr = pd.read_csv(expression_mat_path, index_col=0)
df_expr.head()
```

```
<table>
<thead>
<tr>
<th></th>
<th>EGFR</th>
<th>Ruthenium_1</th>
<th>Ruthenium_2</th>
<th>Ruthenium_3</th>
<th>Ruthenium_4</th>
<th>Ruthenium_5</th>
<th>Ruthenium_6</th>
<th>Ruthenium_7</th>
<th>E-Cadherin</th>
<th>DNA1</th>
<th>CD45</th>
<th>CD68</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_126_X14Y7_1</td>
<td>0.281753</td>
<td>1.319588</td>
<td>0.597380</td>
<td>1.782863</td>
<td>1.757824</td>
<td>1.991857</td>
<td>2.580564</td>
<td>2.287167</td>
<td>1.814309</td>
<td>2.261638</td>
<td>0.44733</td>
<td>0.184805</td>
</tr>
<tr>
<td>BaselTMA_SP41_126_X14Y7_2</td>
<td>0.303016</td>
<td>1.319588</td>
<td>0.597380</td>
<td>1.782863</td>
<td>1.757824</td>
<td>1.991857</td>
<td>2.580564</td>
<td>2.287167</td>
<td>1.517685</td>
<td>1.613060</td>
<td>0.46802</td>
<td>0.080406</td>
</tr>
<tr>
<td>BaselTMA_SP41_126_X14Y7_3</td>
<td>0.252374</td>
<td>1.319588</td>
<td>0.597380</td>
<td>1.782863</td>
<td>1.757824</td>
<td>1.991857</td>
<td>2.580564</td>
<td>2.287167</td>
<td>1.246333</td>
<td>2.138744</td>
<td>0.28499</td>
<td>0.203248</td>
</tr>
<tr>
<td>BaselTMA_SP41_126_X14Y7_4</td>
<td>0.397732</td>
<td>1.306852</td>
<td>0.534496</td>
<td>1.678217</td>
<td>1.757824</td>
<td>1.961430</td>
<td>2.528551</td>
<td>2.183814</td>
<td>1.246333</td>
<td>2.138744</td>
<td>0.28499</td>
<td>0.203248</td>
</tr>
<tr>
<td>BaselTMA_SP41_126_X14Y7_5</td>
<td>0.420522</td>
<td>1.173439</td>
<td>0.597380</td>
<td>1.589303</td>
<td>1.757824</td>
<td>1.991857</td>
<td>2.580564</td>
<td>2.287167</td>
<td>1.246333</td>
<td>2.138744</td>
<td>0.28499</td>
<td>0.203248</td>
</tr>
</tbody>
</table>
```

(continues on next page)
[7]: a_df = ast.Astir(input_expr=df_expr, marker_dict=marker_dict, design=design_df)
    print(a_df)
Astir object, 6 cell types, 4 cell states, 49 cells

1.2 Loading data as np.array or torch.tensor

When the input is Tuple[Union[np.array, torch.tensor]], List[str], List[str], the first element np.array or torch.tensor is the input dataset, the second element List[str] is the title of the columns (the names of proteins) and the third element List[str] is the title of the rows (the name of the cells). The length of the second and third list should be equal to the number of columns and rows of the first array.

[8]: # Load as np.array
    np_expr = df_expr.values
    features = list(df_expr.columns)
    cores = list(df_expr.index)
    a_np = ast.Astir(input_expr=(np_expr, features, cores), marker_dict=marker_dict, design=design_df)
    print(a_np)
Astir object, 6 cell types, 4 cell states, 49 cells

[9]: # Load as torch.tensor
    t_expr = torch.from_numpy(np_expr)
    a_t = ast.Astir(input_expr=(t_expr, features, cores), marker_dict=marker_dict, design=design_df)
1.3 Loading data as SCDataset

When the input is Tuple[SCDataset, SCDataset], the first SCDataset should be the cell type dataset and the second SCDataset should be the cell state dataset.

```python
[10]: type_scd = ast.SCDataset(expr_input=df_expr, marker_dict=marker_dict["cell_types"], include_other_column=True, design=design_df)
state_scd = ast.SCDataset(expr_input=df_expr, marker_dict=marker_dict["cell_states"], include_other_column=False, design=design_df)
a_scd = ast.Astir(input_expr=(type_scd, state_scd))
print(a_scd)
Ashir object, 6 cell types, 4 cell states, 49 cells
```

2. Loading from csv and yaml files

A data reader from_csv_yaml for loading csv and yaml file is provided.

The row of the csv file should represent the information of each single cells and the column of the csv file should represent the expression of each protein in different cells.

```python
[11]: a_csv = ast.from_csv_yaml(csv_input=expression_mat_path, marker_yaml=yaml_marker_path, design_csv=design_mat_path)
print(a_csv)
Astir object, 6 cell types, 4 cell states, 49 cells
```

Some notes:
1. The yaml file at yaml_marker_path contains the marker which maps protein features to cell type/state classes. The format should match the description of **marker dictionary**.
2. from_csv_yaml returns an Astir object.
3. dtype and random_seed are also customizable. dtype is default to torch.float64 and random_seed is default to 1234.

```python
[12]: type(a_csv.get_type_dataset().get_exprs())
```
```
```
torch.Tensor
```

```
[13]: a_csv.get_type_dataset().get_exprs_df().head()
```
```
<table>
<thead>
<tr>
<th>CD20</th>
<th>CD3</th>
<th>CD45</th>
<th>CD68</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_126_X14Y7_1</td>
<td>0.207884</td>
<td>0.000000</td>
<td>0.044733</td>
</tr>
<tr>
<td>BaselTMA_SP41_126_X14Y7_2</td>
<td>0.021506</td>
<td>0.110806</td>
<td>0.046802</td>
</tr>
<tr>
<td>BaselTMA_SP41_126_X14Y7_3</td>
<td>0.008878</td>
<td>0.020617</td>
<td>0.028499</td>
</tr>
<tr>
<td>BaselTMA_SP41_126_X14Y7_4</td>
<td>0.053027</td>
<td>0.060264</td>
<td>0.069053</td>
</tr>
<tr>
<td>BaselTMA_SP41_126_X14Y7_5</td>
<td>0.019127</td>
<td>0.057195</td>
<td>0.233777</td>
</tr>
</tbody>
</table>
```

(continues on next page)
Cytokeratin 14  Cytokeratin 19  Cytokeratin 5
BaselTMA_SP41_126_X14Y7_1  0.134128  0.079956  0.178350
BaselTMA_SP41_126_X14Y7_2  0.026951  0.066922  0.081147
BaselTMA_SP41_126_X14Y7_3  0.023515  0.186294  0.076112
BaselTMA_SP41_126_X14Y7_4  0.114420  0.346273  0.164059
BaselTMA_SP41_126_X14Y7_5  0.055368  0.124407  0.095323

Cytokeratin 7  Cytokeratin 8/18  E-Cadherin
BaselTMA_SP41_126_X14Y7_1  0.043423  0.025526  1.814309
BaselTMA_SP41_126_X14Y7_2  0.032056  0.000000  1.517685
BaselTMA_SP41_126_X14Y7_3  0.081503  0.083311  1.246433
BaselTMA_SP41_126_X14Y7_4  0.131531  0.184603  1.839785
BaselTMA_SP41_126_X14Y7_5  0.038448  0.035371  1.618347

Fibronectin  Her2  Vimentin  pan  Cytokeratin
BaselTMA_SP41_126_X14Y7_1  1.039734  0.483007  0.444140  1.187512
BaselTMA_SP41_126_X14Y7_2  1.147644  0.513386  0.270070  0.749379
BaselTMA_SP41_126_X14Y7_3  0.988906  0.633226  0.233909  1.216521
BaselTMA_SP41_126_X14Y7_4  0.842710  0.709272  0.542362  1.354303
BaselTMA_SP41_126_X14Y7_5  1.073357  0.482330  0.759944  0.629398

3. Loading from a directory of csvs and yaml

The user can also load the data from a directory of csv files with `from_csv_dir_yaml`.

In this case, every csv file should represent the expression data from different samples. A design matrix will be generated automatically.

```
[14]: a_dir = ast.from_csv_dir_yaml(input_dir=expression_dir_path, marker_yaml=yaml_marker_path)
print(a_dir)
```

Astir object, 6 cell types, 4 cell states, 40 cells

Some notes:

1. The yaml file at `yaml_marker_path` contains the marker which maps protein features to cell type/state classes. The format should match the description of *marker dictionary*.

2. `from_csv_dir_yaml` returns an Astir object.

3. `dtype` and `random_seed` are also customizable. `dtype` is default to `torch.float64` and `random_seed` is default to 1234.

```
[15]: type(a_dir.get_type_dataset().get_exprs())
[15]: torch.Tensor

[16]: a_dir.get_type_dataset().get_exprs_df().head()
```

<table>
<thead>
<tr>
<th>CD20</th>
<th>CD3</th>
<th>CD45</th>
<th>CD68</th>
</tr>
</thead>
</table>
| BaselTMA_SP41_126_X14Y7_1 0.207884 0.000000 0.044733 0.184805
| BaselTMA_SP41_126_X14Y7_2 0.021506 0.110806 0.046802 0.080406
| BaselTMA_SP41_126_X14Y7_3 0.081503 0.083311 0.028499 0.203248 |
4. Loading from loom

It is also possible to load the data from a loom file with `from_loompy_yaml`.

```python
[17]: a_loom = ast.from_loompy_yaml(loom_file=expression_loom_path, marker_yaml=yaml_marker_path, protein_name_attr="protein", cell_name_attr="cell_name", batch_name_attr="batch")
print(a_loom)
```

Astir object, 6 cell types, 4 cell states, 100 cells

Some notes:

1. The protein and cell names are taken from `ds.ra[protein_name_attr]` and `ds.ca[cell_name_attr]` respectively if specified, and `ds.ra["protein"]` and `ds.ca["cell_name"]` otherwise.

2. If `batch_name` is specified, the corresponding column of `ds.ca[batch_name_attr]` will be assumed as the batch variable and turned into a design matrix. Otherwise it is taken as `ds.ca["batch"]`

3. The yaml file at `yaml_marker_path` contains the marker which maps protein features to cell type/state classes. The format should match the description of *marker dictionary*.

4. `from_loom_yaml` returns an Astir object.

5. `dtype` and `random_seed` are also customizable. `dtype` is default to `torch.float64` and `random_seed` is default to 1234.

```python
[18]: type(a_loom.get_type_dataset().get_exprs())
```
```python
[18]: torch.Tensor

[19]: a_loom.get_type_dataset().get_exprs_df()

<table>
<thead>
<tr>
<th></th>
<th>CD20</th>
<th>CD3</th>
<th>CD45</th>
<th>CD68</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaselTMA_SP41_44_X2Y7_726</td>
<td>0.080173</td>
<td>0.010469</td>
<td>0.013425</td>
<td>0.149373</td>
</tr>
<tr>
<td>BaselTMA_SP41_Liver_X2Y1_1909</td>
<td>0.051691</td>
<td>0.139216</td>
<td>0.061985</td>
<td>0.140193</td>
</tr>
<tr>
<td>BaselTMA_SP41_231_X6Y6_10_798</td>
<td>0.000000</td>
<td>0.078386</td>
<td>0.144959</td>
<td>0.570016</td>
</tr>
<tr>
<td>BaselTMA_SP41_141_X11Y2_4968</td>
<td>0.039043</td>
<td>0.028426</td>
<td>0.089894</td>
<td>0.089386</td>
</tr>
<tr>
<td>BaselTMA_SP41_141_X11Y2_746</td>
<td>0.079079</td>
<td>0.184354</td>
<td>1.174959</td>
<td>0.297893</td>
</tr>
<tr>
<td>BaselTMA_SP42_25_X3Y2_1178</td>
<td>0.110930</td>
<td>0.022230</td>
<td>0.031842</td>
<td>0.076643</td>
</tr>
<tr>
<td>BaselTMA_SP43_272_X11Y3_460</td>
<td>0.057730</td>
<td>0.124963</td>
<td>0.000000</td>
<td>0.000000</td>
</tr>
<tr>
<td>BaselTMA_SP42_192_X8Y5_2214</td>
<td>0.053447</td>
<td>0.080335</td>
<td>0.031003</td>
<td>0.050570</td>
</tr>
<tr>
<td>BaselTMA_SP41_203_X8Y8_2433</td>
<td>0.028729</td>
<td>0.030617</td>
<td>0.322813</td>
<td>1.180702</td>
</tr>
<tr>
<td>BaselTMA_SP41_249_X3Y9_996</td>
<td>0.228180</td>
<td>0.117249</td>
<td>0.509954</td>
<td>1.180713</td>
</tr>
<tr>
<td>Cytokeratin 14</td>
<td>0.128329</td>
<td>1.577395</td>
<td>0.210581</td>
<td></td>
</tr>
<tr>
<td>Cytokeratin 19</td>
<td>0.208142</td>
<td>0.129807</td>
<td>0.117969</td>
<td></td>
</tr>
<tr>
<td>Cytokeratin 5</td>
<td>0.158847</td>
<td>0.325296</td>
<td>0.13921</td>
<td></td>
</tr>
<tr>
<td>Cytokeratin 7</td>
<td>0.075023</td>
<td>0.294802</td>
<td>0.130921</td>
<td></td>
</tr>
<tr>
<td>Cytokeratin 8/18</td>
<td>0.039844</td>
<td>0.177649</td>
<td>0.129131</td>
<td></td>
</tr>
<tr>
<td>E-Cadherin</td>
<td>0.128341</td>
<td>0.845475</td>
<td>0.036395</td>
<td></td>
</tr>
<tr>
<td>Fibronectin</td>
<td>0.208142</td>
<td>1.777395</td>
<td>2.028177</td>
<td></td>
</tr>
<tr>
<td>Her2</td>
<td>0.051691</td>
<td>0.129807</td>
<td>0.117969</td>
<td></td>
</tr>
<tr>
<td>Vimentin</td>
<td>0.158847</td>
<td>0.325296</td>
<td>0.13921</td>
<td></td>
</tr>
<tr>
<td>(continues on next page)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
5. Loading from anndata

We can read in data from the AnnData format, along with a yaml file containing marker information using the
from_anndata_yaml function. We have temporarily disabled this example while the anndata format is standardized:

```python
[1]: if False:
    a_ann = ast.from_anndata_yaml(anndata_file=expression_anndata_path, marker_yaml=yaml_marker_path,
                                   protein_name="protein",cell_name="cell_name", batch_name="batch")
    print(a_ann)
```

Some notes:

1. The protein and cell names are taken from adata.var[protein_name] and adata.obs[cell_name] respectively if specified, and adata.var_names and adata.obs_names otherwise.

2. If batch_name is specified, the corresponding column of adata.var will be assumed as the batch variable and turned into a design matrix.

3. The yaml file at yaml_marker_path contains the marker which maps protein features to cell type/state classes. The format should match the description of *marker dictionary*.

4. from_anndata_yaml returns an Astir object.

5. dtype and random_seed are also customizable. dtype is default to torch.float64 and random_seed is default to 1234.

```python
[21]: if False:
    type(a_ann.get_type_dataset().get_exprs())
```

```python
[21]: torch.Tensor
```

```python
[22]: if False:
    a_ann.get_type_dataset().get_exprs_df()
```

### 2.2. Tutorials
### Table 2.1

<table>
<thead>
<tr>
<th></th>
<th>CD20</th>
<th>CD3</th>
<th>CD45</th>
<th>CD68</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZTMA208_slide_11_By5x8_1</td>
<td>0.168521</td>
<td>0.090277</td>
<td>0.271871</td>
<td>0.412439</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_2</td>
<td>0.366301</td>
<td>0.352614</td>
<td>0.284034</td>
<td>0.312862</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_3</td>
<td>0.170066</td>
<td>0.103808</td>
<td>0.150791</td>
<td>0.122472</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_4</td>
<td>0.390468</td>
<td>0.2228802</td>
<td>0.219736</td>
<td>0.277622</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_5</td>
<td>0.137789</td>
<td>0.130010</td>
<td>0.105604</td>
<td>1.035280</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_6</td>
<td>0.182926</td>
<td>0.169596</td>
<td>0.270698</td>
<td>0.257178</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_7</td>
<td>0.239257</td>
<td>0.149007</td>
<td>0.351788</td>
<td>0.138080</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_8</td>
<td>0.175299</td>
<td>0.153332</td>
<td>0.215698</td>
<td>0.104709</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_9</td>
<td>0.210541</td>
<td>0.118273</td>
<td>0.146135</td>
<td>0.148164</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_10</td>
<td>0.308899</td>
<td>0.326121</td>
<td>0.224866</td>
<td>0.276182</td>
</tr>
</tbody>
</table>

### Table 2.2

<table>
<thead>
<tr>
<th></th>
<th>Cytokeratin 14</th>
<th>Cytokeratin 19</th>
<th>Cytokeratin 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZTMA208_slide_11_By5x8_1</td>
<td>0.087354</td>
<td>0.155710</td>
<td>0.100308</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_2</td>
<td>0.152354</td>
<td>0.508728</td>
<td>0.028651</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_3</td>
<td>0.292241</td>
<td>0.634366</td>
<td>0.090457</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_4</td>
<td>0.373870</td>
<td>2.212514</td>
<td>0.304824</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_5</td>
<td>0.212105</td>
<td>0.144144</td>
<td>0.074692</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_6</td>
<td>0.224863</td>
<td>1.143546</td>
<td>0.189600</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_7</td>
<td>0.142505</td>
<td>1.415104</td>
<td>0.124484</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_8</td>
<td>0.237387</td>
<td>2.190369</td>
<td>0.264600</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_9</td>
<td>0.362226</td>
<td>1.267224</td>
<td>0.173477</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_10</td>
<td>0.140240</td>
<td>2.032473</td>
<td>0.334358</td>
</tr>
</tbody>
</table>

### Table 2.3

<table>
<thead>
<tr>
<th></th>
<th>Cytokeratin 7</th>
<th>Cytokeratin 8/18</th>
<th>E-Cadherin</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZTMA208_slide_11_By5x8_1</td>
<td>0.000000</td>
<td>0.096674</td>
<td>0.974271</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_2</td>
<td>0.029904</td>
<td>0.749755</td>
<td>2.787740</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_3</td>
<td>0.056627</td>
<td>0.446911</td>
<td>1.927940</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_4</td>
<td>0.000000</td>
<td>1.904837</td>
<td>3.175959</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_5</td>
<td>0.000000</td>
<td>0.000000</td>
<td>1.900182</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_6</td>
<td>0.001542</td>
<td>0.650384</td>
<td>2.580153</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_7</td>
<td>0.001245</td>
<td>1.091975</td>
<td>2.696699</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_8</td>
<td>0.000000</td>
<td>1.457901</td>
<td>2.788996</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_9</td>
<td>0.000000</td>
<td>0.842407</td>
<td>2.950440</td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_10</td>
<td>0.000000</td>
<td>1.503531</td>
<td>2.938590</td>
</tr>
</tbody>
</table>

### Table 2.4

<table>
<thead>
<tr>
<th></th>
<th>Fibronectin</th>
<th>Her2</th>
<th>Vimentin</th>
<th>pan</th>
<th>Cytokeratin</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZTMA208_slide_11_By5x8_1</td>
<td>2.867470</td>
<td>0.552905</td>
<td>2.335253</td>
<td>1.361075</td>
<td></td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_2</td>
<td>2.174494</td>
<td>1.046198</td>
<td>0.285699</td>
<td>2.454543</td>
<td></td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_3</td>
<td>2.979134</td>
<td>1.020157</td>
<td>2.887193</td>
<td>2.590460</td>
<td></td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_4</td>
<td>1.598163</td>
<td>2.269974</td>
<td>0.877098</td>
<td>4.250308</td>
<td></td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_5</td>
<td>2.326346</td>
<td>0.610897</td>
<td>2.882146</td>
<td>0.275225</td>
<td></td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_6</td>
<td>1.891692</td>
<td>1.724237</td>
<td>1.931947</td>
<td>2.994441</td>
<td></td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_7</td>
<td>1.994174</td>
<td>1.796137</td>
<td>0.127125</td>
<td>3.523499</td>
<td></td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_8</td>
<td>1.859896</td>
<td>1.726096</td>
<td>0.106661</td>
<td>4.245234</td>
<td></td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_9</td>
<td>1.852758</td>
<td>2.183716</td>
<td>0.957369</td>
<td>3.098247</td>
<td></td>
</tr>
<tr>
<td>ZTMA208_slide_11_By5x8_10</td>
<td>2.192502</td>
<td>2.312838</td>
<td>1.337983</td>
<td>4.199266</td>
<td></td>
</tr>
</tbody>
</table>
6. Loading model from hdf5

The whole model could be saved with `Astir.save_model(<hdf5_name>)`, and the saved hdf5 file could be used to load a new model. This makes it possible to access the previously trained parameters and assignments without having to train the model again.

```
[23]: # Save a trained model
    hdf5_summary = "./data/a_summary.hdf5"
    a_orig = ast.Astir(input_expr=df_expr, marker_dict=marker_dict, design=design_df)
    a_orig.fit_type(max_epochs=5, n_init=1, n_init_epochs=1)
    a_orig.save_models(hdf5_summary)

    training restart 1/1: 100%|| 1/1 [68.35epochs/s, current loss: 286.9]
    training restart (final): 100%|| 5/5 [90.87epochs/s, current loss: 229.9]
       Maximum epochs reached. More iteration may be needed to complete the training.
       warnings.warn(msg)
    training restart 1/1: 100%|| 1/1 [123.53epochs/s, current loss: 286.9]
    training restart (final): 100%|| 5/5 [91.99epochs/s, current loss: 229.9]
```

```
[24]: # Load a trained model
    a_load = ast.Astir()
    a_load.load_model(hdf5_summary)
    print(a_load)

    Astir object
```

Some notes: 1. In the example, we didn’t load any dataset, which means we can’t get access to the original dataset on which the model was trained. 2. The new model could still be used to predict cell type/status, etc. 3. If the user want to do further operation which requires access to the original dataset, simply load it again when initializing: `a_load = ast.Astir(input_expr=df_expr, marker_dict=marker_dict, design=design_df)`.

2.3 Development

2.3.1 How to render the docs

Install Sphinx

```
$ pip install sphinx
```

From the main project directory `cd` into docs directory

```
$ cd docs
```

Build the existing reStructuredText files

```
$ make html
```

If the above command causes “Could not import extension <extension-name>” pip install them until the build succeeds.
Open astir/docs/html/index.html in your favourite browser either by copying the absolute path in your browser URL bar. If you are using PyCharm editor, you can right click on index.html in the file browser -> Open in Browser -> select your favourite browser

### 2.3.2 How to run nosetests and add a test

#### Running nosetests

**Method 1**

Run one test module at a time

```bash
$ nosetests astir/tests/test_astir.py
$ nosetests astir/tests/models/test_cellstate.py
```

**Method 2**

Run all test modules at once

..code:

```bash
$ nosetests
```

in any project module directory. You might need install the nose package.

#### Adding a unittest

### 2.3.3 Best git practices

The best git practice is to start your own local branch, and commit to your local branch’s remote branch once in awhile. Once your branch is ready to merge into the origin master repo, you want to git merge, pull, and push.

**Git clone and start a new branch**

This is the first step you want to take and won’t have to repeat unless you want to clone on another machine or create a new branch.

```bash
$ git clone https://github.com/camlab-bioml/astir.git
$ git checkout -b <new-branch-name>
```
Update your copy in the repo (git add, commit, push)

You might want to do git commits once in awhile to save your work or create new checkpoint.

$ git add <filename1> <filename2> ... <filename n>  
$ git commit -m "<your-commit-message>"

Additionally push your work in local branch to its remote branch

$ git push origin <my-working-branch-name>

or

$ git push

If you are using the second command make sure that your local branch, called branch-name, is pushing to its remote branch, called origin/branch-name

Update origin/master (git merge, pull)

To update Master remote branch
First, commit and push all your current work to your remote branch
Second, checkout master

$ git checkout master

This changes your working branch to local master.
You can view your current working branch with the following command

$ git branch

$ git merge <branch-to-merge-current-with>

Resolve any merge conflicts you get. Once the merge is complete and all conflicts are resolved
Update the local master branch by

$ git pull origin master

or depending on your setup you may even be able to run

$ git pull

To merge a branch into the current one Again resolve any conflicts
Update remote master by following the steps outlined in Update your copy in the repo

2.3. Development
2.4 astir package

2.4.1 Module contents

class astir.Astir(input_expr=None, marker_dict=None, design=None, random_seed=1234, dtype=torch.float64)
    Bases: object
    Create an Astir object

    Parameters
    • input_expr (Union[pd.DataFrame, Tuple[np.array, List[str], List[str]], Tuple[SCDataset, SCDataset]) – the single cell protein expression dataset
    • marker_dict (Dict[str, Dict[str, str]], optional) – the marker dictionary which maps cell type/state to protein features, defaults to None
    • design (Union[pd.DataFrame, np.array], optional) – the design matrix labeling the grouping of cell, defaults to None
    • random_seed (int, optional) – random seed for parameter initialization, defaults to 1234
    • dtype (torch.dtype, optional) – dtype of data, defaults to torch.float64

    Raises NotClassifiableError – raised if the model is not trainable

    Methods
    assign_celltype_hierarchy([depth]) Get cell type assignment at a specified higher hierarchy according to the hierarchy provided
    diagnostics_cellstate() Run diagnostics on cell state assignments
    diagnostics_celltype([threshold, alpha]) Run diagnostics on cell type assignments
    fit_state([max_epochs, learning_rate, ...]) Run Variational Bayes to infer cell states
    fit_type([max_epochs, learning_rate, ...]) Run Variational Bayes to infer cell types
    get_cellstates() Get cell state activations.
    get_celltype_probabilities() Get the cell assignment probability.
    get_celltypes([threshold, assignment_type]) Get the most likely cell type
    get_hierarchy_dict() Get the dictionary for cell type hierarchical structure.
    get_state_dataset() Get the SCDataset for cell state training.
    get_state_losses() Getter for losses
    get_state_model() Get the trained CellStateModel.
    get_state_run_info() Get the run information (i.e. max_epochs, learning_rate, batch_size).
    get_type_dataset() Get the SCDataset for cell type training.
    get_type_losses() Get the final losses of the type model.
    get_type_model() Get the trained CellTypeModel.
    get_type_run_info() Get the run information (i.e. max_epochs, learning_rate).
    load_model(hdf5_name) Load model from hdf5 file
    normalize([percentile_lower, percentile_upper]) Normalize the expression data
    predict_cellstates([dset]) Get the prediction cell state activations on a dataset on an existing model

continues on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict_celltypes([dset])</td>
<td>Predict the probabilities of different cell type assignments.</td>
</tr>
<tr>
<td>save_models([hdf5_name])</td>
<td>Save the summary of this model to an hdf5 file.</td>
</tr>
<tr>
<td>state_to_csv(output_csv)</td>
<td>Writes state assignment output from training state model in csv file</td>
</tr>
<tr>
<td>type_clustermap([plot_name, threshold, ...])</td>
<td>Save the heatmap of protein content in cells with cell types labeled.</td>
</tr>
<tr>
<td>type_to_csv(output_csv[, threshold, ...])</td>
<td>Save the cell type assignment to a csv file.</td>
</tr>
</tbody>
</table>

assign_celltype_hierarchy(depth=1)

Get cell type assignment at a specified higher hierarchy according to the hierarchy provided in the dictionary.

Parameters
- **depth** (int, optional) – the depth of hierarchy to assign probability to, defaults to 1

Raises **Exception** – raised when the dictionary for hierarchical structure is not provided or the model hasn’t been trained.

Returns probability assignment of cell type at a superstructure.

Return type pd.DataFrame

diagnostics_cellstate()

Run diagnostics on cell state assignments

This performs a basic test by comparing the correlation values between all marker genes and all non marker genes. It detects where the non marker gene has higher correlation values than the smallest correlation values of marker genes.

1. Get correlations between all cell states and proteins
2. For each cell state \(c\), get the smallest correlation with marker \(g\)
3. For each cell state \(c\) and its non marker \(g\), find any correlation that is bigger than those smallest correlation for \(c\).
4. Any \(c\) and \(g\) pairs found in step 3 will be included in the output of `Astir.diagnostics_cellstate()`, including an explanation.

Return type DataFrame

Returns diagnostics

diagnostics_celltype(threshold=0.5, alpha=0.01)

Run diagnostics on cell type assignments

This performs a basic test that cell types express their markers at higher levels than in other cell types. This function performs the following steps:

1. Iterates through every cell type and every marker for that cell type
2. Given a cell type \(c\) and marker \(g\), find the set of cell types \(D\) that don’t have \(g\) as a marker
3. For each cell type \(d\) in \(D\), perform a t-test between the expression of marker \(g\) in \(c\) vs \(d\)
4. If \(g\) is not expressed significantly higher (at significance \(alpha\)), output a diagnostic explaining this for further investigation.
Parameters

- **threshold** (float) – The threshold at which cell types are assigned (see `get_celltypes`)
- **alpha** (float) – The significance threshold for t-tests for determining over-expression

Return type: DataFrame

Returns: Either a `pd.DataFrame` listing cell types whose markers aren’t expressed significantly higher.

```
fit_state(max_epochs=50, learning_rate=0.001, batch_size=128, delta_loss=0.001, n_init=5, n_init_epochs=5, delta_loss_batch=10, const=2, dropout_rate=0, batch_norm=False)
```

Run Variational Bayes to infer cell states

Parameters

- **max_epochs** (int) – number of epochs, defaults to 100
- **learning_rate** (float) – the learning rate, defaults to 1e-2
- **n_init** (int) – the number of initial parameters to compare, defaults to 5
- **delta_loss** (float) – stops iteration once the loss rate reaches delta_loss, defaults to 0.001
- **delta_loss_batch** (int) – the batch size to consider delta loss, defaults to 10

Return type: None

```
fit_type(max_epochs=50, learning_rate=0.001, batch_size=128, delta_loss=0.001, n_init=5, n_init_epochs=5)
```

Run Variational Bayes to infer cell types

Parameters

- **max_epochs** (int) – maximum number of epochs to train
- **learning_rate** (float) – ADAM optimizer learning rate
- **batch_size** (int) – minibatch size
- **delta_loss** (float) – stops iteration once the loss rate reaches delta_loss, defaults to 0.001
- **n_inits** – number of random initializations

Return type: None

```
get_cellstates()
```

Get cell state activations. It returns the rescaled activations, values between 0 and 1

Returns: state assignments

Return type: `pd.DataFrame`

```
get_celltype_probabilities()
```

Get the cell assignment probability.

Returns: `self.assignments`

Return type: `pd.DataFrame`

```
get_celltypes(threshold=0.7, assignment_type='threshold')
```

Get the most likely cell type
A cell is assigned to a cell type if the probability is greater than threshold. If no cell types have a probability higher than threshold, then “Unknown” is returned

**Parameters**

- **threshold** (float) – the probability threshold above which a cell is assigned to a cell type
- **assignment_type** (str) – See `astir.CellTypeModel.get_celltypes()` for full documentation

**Return type** DataFrame

**Returns** a data frame with most likely cell types for each

`get_hierarchy_dict()`

Get the dictionary for cell type hierarchical structure.

- **Returns** `self._hierarchy_dict`
- **Return type** `Dict[str, List[str]]`

`get_state_dataset()`

Get the `SCDataset` for cell state training.

- **Return type** `SCDataset`
- **Returns** `self._state_dset`

`get_state_losses()`

Getter for losses

- **Returns** a numpy array of losses for each training iteration the model runs
- **Return type** `np.array`

`get_state_model()`

Get the trained `CellStateModel`.

- **Raises** `Exception` – raised when this function is called before the model is trained.
- **Return type** `CellStateModel`
- **Returns** `self._state_ast`

`get_state_run_info()`

Get the run information (i.e. `max_epochs`, `learning_rate`, `batch_size`, `delta_loss`, `n_init`, `n_init_epochs`, `delta_loss_batch`) of the cell state training.

- **Raises** `Exception` – raised when this function is called before the model is trained.
- **Return type** `Dict[str, Union[int, float]]`
- **Returns** `self._state_run_info`

`get_type_dataset()`

Get the `SCDataset` for cell type training.

- **Return type** `SCDataset`
- **Returns** `self._type_dset`

`get_type_losses()`

Get the final losses of the type model.
astir

- **Returns**: `self.losses`
- **Return type**: `np.array`

**get_type_model()**

Get the trained `CellTypeModel`.

- **Raises**: `Exception` – raised when this function is called before the model is trained.
- **Return type**: `CellTypeModel`
- **Returns**: `self._type_ast`

**get_type_run_info()**

Get the run information (i.e. `max_epochs`, `learning_rate`, `batch_size`, `delta_loss`, `n_init`, `n_init_epochs`) of the cell type training.

- **Raises**: `Exception` – raised when this function is called before the model is trained.
- **Return type**: `Dict[typing.Union[int, float]]`
- **Returns**: `self._type_run_info`

**load_model(hdf5_name)**

Load model from hdf5 file.

- **Parameters**:
  - `hdf5_name` (**str**): the full path to file
- **Return type**: `None`

**normalize(percentile_lower=1, percentile_upper=99)**

Normalize the expression data.

This performs a two-step normalization: 1. A \(\log(1+x)\) transformation to the data 2. Winsorizes to \((\text{percentile_lower}, \text{percentile_upper})\)

- **Parameters**:
  - `percentile_lower` (**int**): Lower percentile for winsorization
  - `percentile_upper` (**int**): Upper percentile for winsorization
- **Return type**: `None`

**predict_cellstates(dset=None)**

Get the prediction cell state activations on a dataset on an existing model.

- **Parameters**:
  - `dset` (**pd.DataFrame**, optional): the dataset to predict cell state activations, default to None
- **Return type**: `DataFrame`
- **Returns**: the prediction of cell state activations

**predict_celltypes(dset=None)**

Predict the probabilities of different cell type assignments.

- **Parameters**:
  - `dset` (**pd.DataFrame**, optional): the single cell protein expression dataset to predict, defaults to None
- **Raises**: `Exception` – when the type model is not trained when this function is called
- **Return type**: `pd.DataFrame`
- **Returns**: the probabilities of different cell type assignments
save_models(hdf5_name='astir_summary.hdf5')
    Save the summary of this model to an hdf5 file.
    Parameters hdf5_name (str) – name of the output hdf5 file, default to “astir_summary.hdf5”
    Raises Exception – raised when this function is called before the model is trained.
    Return type None

state_to_csv(output_csv)
    Writes state assignment output from training state model in csv file
    Parameters output_csv (str, required) – path to output csv
    Return type None

type_clustermap(plot_name='celltype_protein_cluster.png', threshold=0.7, figsize=(7, 5), prob_assign=None)
    Save the heatmap of protein content in cells with cell types labeled.
    Parameters
        • plot_name (str, optional) – name of the plot, extension(e.g. .png or .jpg) is needed, defaults to “celltype_protein_cluster.png”
        • threshold (float, optional) – the probability threshold above which a cell is assigned to a cell type, defaults to 0.7
    Return type None

type_to_csv(output_csv, threshold=0.7, assignment_type='threshold')
    Save the cell type assignment to a csv file.
    Parameters
        • output_csv (str) – name for the output .csv file
        • assignment_type (str) – See astir.CellTypeModel.get_celltypes() for full documentation
    Return type None

2.5 astir.data package

2.5.1 Module contents

Classes:

SCDataset(expr_input, marker_dict, ...[, ...])
    Container for single-cell proteomic data in the form of a pytorch dataset

Functions:

from_anndata_yaml(anndata_file, marker_yaml)
    Create an Astir object from an anndata.Anndata file and a

from_csv_dir_yaml(input_dir, marker_yaml[, ...])
    Create an Astir object a directory containing multiple csv files

continues on next page
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from_csv_yaml</code></td>
<td>Create an Astir object from an expression CSV and marker YAML</td>
</tr>
<tr>
<td><code>from_loompy_yaml</code></td>
<td>Create an Astir object from a loom file and a marker YAML</td>
</tr>
</tbody>
</table>

**class astir.data.SCDataset**

```
class astir.data.SCDataset(expr_input, marker_dict[, ...])
```

Container for single-cell proteomic data in the form of a pytorch dataset

**Parameters**

- **expr_input** *(Union[DataFrame, Tuple[Union[array, Tensor], List[str], List[str]])* – Input expression data. See details: `expr_input` is either a `pd.DataFrame` or a three-element tuple. When it is `pd.DataFrame`, its index and column should indicate the cell name and feature name of the dataset; when it is a three-element tuple, it should be in the form of `Tuple[Union[np.array, torch.Tensor], List[str], List[str]]` and its first element should be the actual dataset as either `np.array` or `torch.tensor`, the second element should be a list containing the name of the columns or the names of features, the third element should be a list containing the name of the indices or the names of the cells.

- **marker_dict** *(Dict[str, List[str]])* – Marker dictionary containing cell type and information. See details: The dictionary maps the name of cell type/state to protein features.

- **design** *(Union[DataFrame, array, None]*) – A design matrix

- **include_other_column** *(bool)* – Should an additional ‘other’ column be included?

- **dtype** *(dtype)* – torch datatype of the model

**Methods:**

- `get_cell_names()` Get the cell names.
- `get_classes()` Get the cell types/states.
- `get_design()` Get the design matrix.
- `get_dtype()` Get the dtype of the `SCDataset`.
- `get_exprs()` Return the expression data as a `torch.Tensor`.
- `get_exprs_df()` Return the expression data as a `pandas.DataFrame`.
- `get_features()` Get the features (proteins).
- `get_marker_mat()` Return the marker matrix as a `torch.Tensor`.
- `get_mu()` Get the mean expression of each protein as a `torch.Tensor`.
- `get_mu_init()` Intelligent initialization for mu parameters
- `get_n_cells()` Get the number of cells: either the number of cell types or cell states.
- `get_n_classes()` Get the number of ‘classes’: either the number of cell types or cell states.
- `get_n_features()` Get the number of features (proteins).
- `get_sigma()` Get the standard deviation of each protein
- `normalize()` Normalize the expression data
- `rescale()` Normalize the expression data

**Returns**

- `return self._cell_names`
Return type  List[str]

**get_classes()**
Get the cell types/states.

**Returns**  return self._classes

**Return type**  List[str]

**get_design()**
Get the design matrix.

**Returns**  return self._design

**Return type**  torch.Tensor

**get_dtype()**
Get the dtype of the *SCDataset*.

**Returns**  self._dtype

**Return type**  torch.dtype

**get_exprs()**
Return the expression data as a *torch.Tensor*.

**Return type**  Tensor

**get_exprs_df()**
Return the expression data as a *pandas.DataFrame*.

**Return type**  DataFrame

**get_features()**
Get the features (proteins).

**Returns**  return self._m_features

**Return type**  List[str]

**get_marker_mat()**
Return the marker matrix as a *torch.Tensor*.

**Return type**  Tensor

**get_mu()**
Get the mean expression of each protein as a *torch.Tensor*.

**Return type**  Tensor

**get_mu_init**(n_putative_cells=10)  
Intelligent initialization for mu parameters

See manuscript for details

**Parameters**  n_putative_cells (int) – Number of cells to guess as given cell type

**Return type**  ndarray

**get_n_cells()**
Get the number of cells: either the number of cell types or cell states.

**Return type**  int

**get_n_classes()**
Get the number of ‘classes’: either the number of cell types or cell states.

**Return type**  int
`get_n_features()`  
Get the number of features (proteins).  
Return type  int  

`get_sigma()`  
Get the standard deviation of each protein  
Return type  Tensor  
Returns  standard deviation of each protein  

`normalize(percentile_lower=0, percentile_upper=99.9, cofactor=5.0)`  
Normalize the expression data  
This performs a two-step normalization: 1. A \( \log(1+x) \) transformation to the data 2. Winsorizes to \((\text{percentile}_{\text{lower}}, \text{percentile}_{\text{upper}})\)  
Parameters  
• **percentile_lower** (float) – the lower bound percentile for winsorization, defaults to 0  
• **percentile_upper** – the upper bound percentile for winsorization, defaults to 99.9  
• **cofactor** (float) – a cofactor constant, defaults to 5.0  
Return type  None  

`rescale()`  
Normalize the expression data.  
Return type  None  

`astir.data.from_anndata_yaml(anndata_file, marker_yaml, protein_name=None, cell_name=None, batch_name='batch', create_design_mat=True, random_seed=1234, dtype=torch.float64)`  
Create an Astir object from an anndata.Anndata file and a marker yaml  
Parameters  
• **anndata_file** (str) – Path to an anndata.Anndata h5py file  
• **marker_yaml** (str) – Path to input YAML file containing marker gene information. Should include cell_type and cell_state entries. See documentation.  
• **protein_name** (Optional[str]) – The column of `adata.var` containing protein names. If this is none, defaults to `adata.var_names`  
• **cell_name** (Optional[str]) – The column of `adata.obs` containing cell names. If this is none, defaults to `adata.obs_names`  
• **batch_name** (str) – The column of `adata.obs` containing batch names. A design matrix will be built using this (if present) using a one-hot encoding to control for batch, defaults to ‘batch’  
• **create_design_mat** (bool) – Determines whether a design matrix is created. Defaults to True.  
• **random_seed** (int) – The random seed to be used to initialize variables, defaults to 1234  
• **dtype** (dtype) – datatype of the model parameters, defaults to torch.float64  
Return type  Any
Returns An object of class astir_bash.py.Astir using data imported from the loom files

astir.data.from_csv_dir_yaml(input_dir, marker_yaml, create_design_mat=True, random_seed=1234, dtype=torch.float64)

Create an Astir object a directory containing multiple csv files

Parameters

- **input_dir** (str) – Path to a directory containing multiple CSV files, each in the format expected by from_csv_yaml
- **marker_yaml** (str) – Path to input YAML file containing marker gene information. Should include cell_type and cell_state entries. See documentation.
- **design_csv** – Path to design matrix as a CSV. Rows should be cells, and columns covariates. First column is cell identifier, and additional column names are covariate identifiers
- **create_design_mat** (bool) – Determines whether a design matrix is created. Defaults to True.
- **random_seed** (int) – The random seed to be used to initialize variables, defaults to 1234
- **dtype** (dtype) – datatype of the model parameters, defaults to torch.float64

Return type Any

astir.data.from_csv_yaml(csv_input, marker_yaml, design_csv=None, create_design_mat=True, random_seed=1234, dtype=torch.float64)

Create an Astir object from an expression CSV and marker YAML

Parameters

- **csv_input** (str) – Path to input csv containing expression for cells (rows) by proteins (columns). First column is cell identifier, and additional column names are gene identifiers.
- **marker_yaml** (str) – Path to input YAML file containing marker gene information. Should include cell_type and cell_state entries. See documentation.
- **design_csv** (Optional[str]) – Path to design matrix as a CSV. Rows should be cells, and columns covariates. First column is cell identifier, and additional column names are covariate identifiers.
- **create_design_mat** (bool) – Determines whether a design matrix is created. Defaults to True.
- **random_seed** (int) – The random seed to be used to initialize variables, defaults to 1234
- **dtype** (dtype) – datatype of the model parameters, defaults to torch.float64

Return type Any

astir.data.from_loompy_yaml(loom_file, marker_yaml, protein_name_attr='protein', cell_name_attr='cell_name', batch_name_attr='batch', create_design_mat=True, random_seed=1234, dtype=torch.float64)

Create an Astir object from a loom file and a marker yaml

Parameters

- **loom_file** (str) – Path to a loom file, where rows correspond to proteins and columns to cells
- **marker_yaml** (str) – Path to input YAML file containing marker gene information. Should include cell_type and cell_state entries. See documentation.
• **protein_name_attr** (str) – The attribute (key) in the row attributes that identifies the protein names (required to match with the marker gene information), defaults to protein

• **cell_name_attr** (str) – The attribute (key) in the column attributes that identifies the name of each cell, defaults to cell_name

• **batch_name_attr** (str) – The attribute (key) in the column attributes that identifies the batch. A design matrix will be built using this (if present) using a one-hot encoding to control for batch, defaults to batch

• **create_design_mat** (bool) – Determines whether a design matrix is created. Defaults to True.

• **random_seed** (int) – The random seed to be used to initialize variables, defaults to 1234

• **dtype** (dtype) – datatype of the model parameters, defaults to torch.float64

**Return type** Any

**Returns** An object of class `astir_bash.py.Astir` using data imported from the loom files

### 2.6 astir.models package

#### 2.6.1 Module contents

**Classes:**

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AstirModel</strong></td>
<td>Abstract class to perform statistical inference to assign.</td>
</tr>
<tr>
<td><strong>CellStateModel</strong></td>
<td>Class to perform statistical inference to on the activation</td>
</tr>
<tr>
<td><strong>CellTypeModel</strong></td>
<td>Class to perform statistical inference to assign cells to cell types.</td>
</tr>
<tr>
<td><strong>StateRecognitionNet</strong></td>
<td>State Recognition Neural Network to get mean of z and standard deviation of z.</td>
</tr>
<tr>
<td><strong>TypeRecognitionNet</strong></td>
<td>Type Recognition Neural Network.</td>
</tr>
</tbody>
</table>

**class** `astir.models.AstirModel(dset, random_seed, dtype[, device])`

**Bases:** object

Abstract class to perform statistical inference to assign. This module is the super class of `CellTypeModel` and `CellStateModel` and is not supposed to be instantiated.

**Methods:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fit</strong>(max_epochs, learning_rate, batch_size,...)</td>
<td>Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times</td>
</tr>
<tr>
<td><strong>get_assignment</strong>()</td>
<td>Get the final assignment of the dataset.</td>
</tr>
<tr>
<td><strong>get_data()</strong></td>
<td>Get model data</td>
</tr>
<tr>
<td><strong>get_losses()</strong></td>
<td>Getter for losses.</td>
</tr>
<tr>
<td><strong>get_scdataset()</strong></td>
<td>Getter for the <code>SCDataset</code>.</td>
</tr>
<tr>
<td><strong>get_variables()</strong></td>
<td>Returns all variables</td>
</tr>
<tr>
<td><strong>is_converged()</strong></td>
<td>Returns True if the model converged</td>
</tr>
</tbody>
</table>

**fit**(max_epochs, learning_rate, batch_size, delta_loss, delta_loss_batch, msg)

Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times.
number of times

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_assignment()</td>
<td>DataFrame</td>
<td>Get the final assignment of the dataset.</td>
</tr>
<tr>
<td>get_data()</td>
<td>Dict[str, Tensor]</td>
<td>Get model data</td>
</tr>
<tr>
<td>get_losses()</td>
<td>Tensor</td>
<td>Getter for losses.</td>
</tr>
<tr>
<td>get_scdataset()</td>
<td>SCDataset</td>
<td>Getter for the SCDataset.</td>
</tr>
<tr>
<td>get_variables()</td>
<td>Dict[str, Tensor]</td>
<td>Returns all variables</td>
</tr>
<tr>
<td>is_converged()</td>
<td>bool</td>
<td>Returns True if the model converged.</td>
</tr>
</tbody>
</table>

### Class: astir.models.CellStateModel

**Class to perform statistical inference to on the activation of states (pathways) across cells**

**Parameters**

- **dset (Optional[SCDataset])** – the input gene expression dataset, defaults to None
- **const (int)** – See parameter const in astir.models.StateRecognitionNet(), defaults to 2
- **dropout_rate (float)** – See parameter dropout_rate in astir.models.StateRecognitionNet(), defaults to 0
- **batch_norm (bool)** – See parameter batch_norm in astir.models.StateRecognitionNet(), defaults to False
- **random_seed (int)** – the random seed number to reproduce results, defaults to 42
• **dtype** (dtype) – torch datatype to use in the model, defaults to torch.float64
• **device** (device) – torch.device’s cpu or gpu, defaults to torch.device(“cpu”)

Methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>diagnostics()</code></td>
<td>Run diagnostics on cell type assignments</td>
</tr>
<tr>
<td><code>fit(max_epochs, learning_rate, batch_size, ...)</code></td>
<td>Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times</td>
</tr>
<tr>
<td><code>get_correlations()</code></td>
<td>Returns a C (# of pathways) X G (# of proteins) matrix where each element represents the correlation value of the pathway and the protein</td>
</tr>
<tr>
<td><code>get_final_mu_z(new_dset)</code></td>
<td>Returns the mean of the predicted z values for each core</td>
</tr>
<tr>
<td><code>get_recognet()</code></td>
<td>Getter for the recognition net</td>
</tr>
<tr>
<td><code>load_hdf5(hdf5_name)</code></td>
<td>Initializes Cell State Model from a hdf5 file type</td>
</tr>
</tbody>
</table>

`diagnostics()`

Run diagnostics on cell type assignments

See astir.Astir.diagnostics_cellstate() for full documentation

Return type DataFrame

`fit(max_epochs=50, learning_rate=0.001, batch_size=128, delta_loss=0.001, delta_loss_batch=10, msg='')`

Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times

Parameters

• **max_epochs** (int) – number of train loop iterations, defaults to 50
• **learning_rate** (float) – the learning rate, defaults to 0.01
• **batch_size** (int) – the batch size, defaults to 128
• **delta_loss** (float) – stops iteration once the loss rate reaches delta_loss, defaults to 0.001
• **delta_loss_batch** (int) – the batch size to consider delta loss, defaults to 10
• **msg** (str) – iterator bar message, defaults to empty string

Return type None

`get_correlations()`

Returns a C (# of pathways) X G (# of proteins) matrix where each element represents the correlation value of the pathway and the protein

Return type array

Returns matrix of correlation between all pathway and protein pairs.

`get_final_mu_z(new_dset=None)`

Returns the mean of the predicted z values for each core

Parameters **new_dset** (Optional[SCDataset]) – returns the predicted z values of this dataset on the existing model. If None, it predicts using the existing dataset, defaults to None

Return type Tensor
Returns the mean of the predicted z values for each core

get_recognet()
    Getter for the recognition net

Return type StateRecognitionNet

Returns the recognition net

load_hdf5(hdf5_name)
    Initializes Cell State Model from a hdf5 file type

Parameters hdf5_name (str) – file path

Return type None

class astir.models.CellTypeModel (dset=None, random_seed=1234, dtype=torch.float64, device=device(type='cpu'))
Bases: astir.models.abstract.AstirModel

Class to perform statistical inference to assign cells to cell types.

Parameters

• dset (Optional[SCDataset]) – the input gene expression dataframe
• random_seed (int) – the random seed for parameter initialization, defaults to 1234
• dtype (dtype) – the data type of parameters, should be the same as dset, defaults to torch.float64

Methods:

diagnostics(cell_type_assignments, alpha) Run diagnostics on cell type assignments
fit([max_epochs, learning_rate, batch_size, ...]) Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times
get_celltypes([threshold, assignment_type, ...]) Get the most likely cell types.
get_recognet() Getter for the recognition net.
load_hdf5(hdf5_name) Initializes Cell Type Model from a hdf5 file type
plot_clustermap([plot_name, threshold, ...]) Save the heatmap of protein content in cells with cell types labeled.
predict(new_dset) Feed new_dset to the recognition net to get a prediction.

diagnostics (cell_type_assignments, alpha)
    Run diagnostics on cell type assignments
    See astir.Astir.diagnostics_celltype() for full documentation

Return type DataFrame

fit(max_epochs=50, learning_rate=0.001, batch_size=128, delta_loss=0.001, delta_loss_batch=10, msg='')
    Runs train loops until the convergence reaches delta_loss for delta_loss_batch sizes or for max_epochs number of times

Parameters

• max_epochs (int) – number of train loop iterations, defaults to 50
• learning_rate (float) – the learning rate, defaults to 0.01
• **batch_size** (int) – the batch size, defaults to 128
• **delta_loss** (float) – stops iteration once the loss rate reaches delta_loss, defaults to 0.001
• **delta_loss_batch** (int) – the batch size to consider delta loss, defaults to 10
• **msg** (str) – iterator bar message, defaults to empty string

Return type None

**get_celltypes**(threshold=0.7, **assignment_type**='threshold', prob_assign=None)
Get the most likely cell types. A cell is assigned to a cell type if the probability is greater than threshold. If no cell types have a probability higher than threshold, then “Unknown” is returned.

Parameters
• **assignment_type** (str) – either 'threshold' or 'max'. If threshold, type assignment is based on whether the probability threshold is above prob_assignment. If 'max', type assignment is based on the max probability value or “unknown” if there are multiple max probabilities. Defaults to ‘threshold’.
• **threshold** (float) – the probability threshold above which a cell is assigned to a cell type, defaults to 0.7

Return type DataFrame

Returns a data frame with most likely cell types for each

**get_recognet**()
Get for the recognition net.

Return type **TypeRecognitionNet**

Returns the trained recognition net

**load_hdf5**(hdf5_name)
Initializes Cell Type Model from a hdf5 file type

Parameters hdf5_name (str) – file path

Return type None

**plot_clustermap**(plot_name='celltype_protein_cluster.png', threshold=0.7, figsize=(7.0, 5.0), prob_assign=None)
Save the heatmap of protein content in cells with cell types labeled.

Parameters
• **plot_name** (str) – name of the plot, extension(e.g. .png or .jpg) is needed, defaults to “celltype_protein_cluster.png”
• **threshold** (float) – the probability threshold above which a cell is assigned to a cell type, defaults to 0.7
• **figsize** (Tuple[float, float]) – the size of the figure, defaults to (7.0, 5.0)

Return type None

**predict**(new_dset)
Feed new_dset to the recognition net to get a prediction.

Parameters new_dset (DataFrame) – the dataset to be predicted

Return type array

Returns the resulting cell type assignment
class astir.models.StateRecognitionNet(C, G, const=2, dropout_rate=0, batch_norm=False)
    Bases: torch.nn.modules.module.Module

State Recognition Neural Network to get mean of z and standard deviation of z. The neural network architecture looks like this: G $\rightarrow$ const $\times$ C $\rightarrow$ const $\times$ C $\rightarrow$ G (for mu) or $\rightarrow$ G (for std). With batch normal layers after each activation output layers and dropout activation units

Parameters

- C (int) – the number of pathways
- G (int) – the number of proteins
- const (int) – the size of the hidden layers are const times proportional to C, defaults to 2
- dropout_rate (float) – the dropout rate, defaults to 0
- batch_norm (bool) – apply batch normal layers if True, defaults to False

Methods:

forward(x) One forward pass of the StateRecognitionNet

Attributes:

forward(x)
One forward pass of the StateRecognitionNet

Parameters x (Tensor) – the input to the recognition network model

Return type Tuple[Tensor, Tensor]

Returns the value from the output layer of the network

training: bool

class astir.models.TypeRecognitionNet(C, G, hidden_size=20)
    Bases: torch.nn.modules.module.Module

Type Recognition Neural Network.

Parameters

- C (int) – number of classes
- G (int) – number of features
- hidden_size (int) – size of hidden layers, defaults to 10

Methods:

forward(x) One forward pass.

Attributes:

forward(x)
One forward pass.

Parameters x (Tensor) – the input vector
Return type  Tensor
Returns the calculated cost value

training: bool

2.7 Indices and tables

• genindex
• modindex
• search
a

astir.data, 39
astir.models, 44
astir
INDEX

A
assign_celltype_hierarchy() (astir.Astir method), 35
Astir (class in astir), 34
astir.data module, 39
astir.models module, 44
AstirModel (class in astir.models), 44

C
CellStateModel (class in astir.models), 45
CellTypeModel (class in astir.models), 47

D
diagnostics() (astir.models.CellStateModel method), 46
diagnostics() (astir.models.CellTypeModel method), 47
diagnostics_cellstate() (astir.Astir method), 35
diagnostics_celltype() (astir.Astir method), 35

F
fit() (astir.models.AstirModel method), 44
fit() (astir.models.CellStateModel method), 46
fit() (astir.models.CellTypeModel method), 47
fit_state() (astir.Astir method), 36
fit_type() (astir.Astir method), 36
forward() (astir.models.StateRecognitionNet method), 49
forward() (astir.models.TypeRecognitionNet method), 49
from_anndata_yaml() (in module astir.data), 42
from_csv_dir_yaml() (in module astir.data), 43
from_csv_yaml() (in module astir.data), 43
from_loompy_yaml() (in module astir.data), 43

G
get_assignment() (astir.models.AstirModel method), 45
get_cell_names() (astir.data.SCDataset method), 40
get_cellstates() (astir.Astir method), 36
get_celltype_probabilities() (astir.Astir method), 36
get_celltypes() (astir.Astir method), 36
get_celltypes() (astir.models.CellTypeModel method), 48
get_classes() (astir.data.SCDataset method), 41
get_correlations() (astir.models.CellStateModel method), 46
get_data() (astir.models.AstirModel method), 45
get_design() (astir.data.SCDataset method), 41
get_dtype() (astir.data.SCDataset method), 41
get_exprs() (astir.data.SCDataset method), 41
get_exprs_df() (astir.data.SCDataset method), 41
get_features() (astir.data.SCDataset method), 41
get_final_mu_z() (astir.models.CellStateModel method), 46
get_hierarchy_dict() (astir.Astir method), 37
get_losses() (astir.models.AstirModel method), 45
get_marker_mat() (astir.data.SCDataset method), 41
get_mu() (astir.data.SCDataset method), 41
get_mu_init() (astir.data.SCDataset method), 41
get_n_cells() (astir.data.SCDataset method), 41
get_n_classes() (astir.data.SCDataset method), 41
get_n_features() (astir.data.SCDataset method), 41
get_recognet() (astir.models.CellStateModel method), 47
get_recognet() (astir.models.CellTypeModel method), 48
get_scdataset() (astir.models.AstirModel method), 45
get_sigma() (astir.data.SCDataset method), 42
get_state_dataset() (astir.Astir method), 37
get_state_losses() (astir.Astir method), 37
get_state_model() (astir.Astir method), 37
get_state_run_info() (astir.Astir method), 37
get_type_dataset() (astir.Astir method), 37
get_type_losses() (astir.Astir method), 37
get_type_model() (astir.Astir method), 38
get_type_run_info() (astir.Astir method), 38
get_variables() (astir.models.AstirModel method), 45
is_converged() \textit{(astir.models.AstirModel method)}, 45

load_hdf5() \textit{(astir.models.CellStateModel method)}, 47
load_hdf5() \textit{(astir.models.CellTypeModel method)}, 48
load_model() \textit{(astir.Astir method)}, 38

module
\begin{verbatim}
astir.data, 39
astir.models, 44
\end{verbatim}

normalize() \textit{(astir.Astir method)}, 38
normalize() \textit{(astir.data.SCDataset method)}, 42

plot_clustermap() \textit{(astir.models.CellTypeModel method)}, 48
predict() \textit{(astir.models.CellTypeModel method)}, 48
predict_cellstates() \textit{(astir.Astir method)}, 38
predict_celltypes() \textit{(astir.Astir method)}, 38

rescale() \textit{(astir.data.SCDataset method)}, 42

save_models() \textit{(astir.Astir method)}, 38
SCDataset \textit{(class in astir.data)}, 40
state_to_csv() \textit{(astir.Astir method)}, 39
StateRecognitionNet \textit{(class in astir.models)}, 48

training \textit{(astir.models.StateRecognitionNet attribute)}, 49
training \textit{(astir.models.TypeRecognitionNet attribute)}, 50
type_clustermap() \textit{(astir.Astir method)}, 39
type_to_csv() \textit{(astir.Astir method)}, 39
TypeRecognitionNet \textit{(class in astir.models)}, 49